Using Association Rule Mining to Predict Opponent Deck Content in Android: Netrunner

Nick Sephton*, Peter I. Cowling*, Sam Devlin*, Victoria J. Hodge*, and Nicholas H. Slaven†
*York Centre for Complex Systems Analysis, Department of Computer Science, University of York, United Kingdom
Email: njs523@york.ac.uk, peter.cowling@york.ac.uk, sam.devlin@york.ac.uk, victoria.hodge@york.ac.uk
†Stainless Games, Isle of Wight, United Kingdom
Email: nicks@stainlessgames.com

***Abstract*—As part of their design, card games often include information that is hidden from opponents and represents a strategic advantage if discovered. A player that can discover this information will be able to alter their strategy based on the nature of that information, and therefore become a more competent opponent. In this paper, we employ association rule-mining techniques for predicting item multisets, and show them to be effective in predicting the content of Netrunner decks. We then apply different modifications based on heuristic knowledge of the Netrunner game, and show the effectiveness of techniques which consider this knowledge during rule generation and prediction.**

I. Introduction

A variety of games often include incomplete or hidden information as a form of challenge to the players, indeed most such games would be far more trivial if such an element was excluded. Card games in which players bring decks of their own construction to play are now relatively common place, and are represented both in physical card gaming (e.g. Magic: The Gathering[1]), and in digital gaming (e.g. Blizzard Entertainment's Hearthstone[2]). In such games, knowledge of the content of an opponent's deck represents a potentially powerful strategic knowledge which can be exploited to significant advantage. This is true of competition outside of the game domain also, as being able to adequately predict the strategy of a potential competitor will likely give significant advantage.

In this paper we consider a deck of cards to be a multiset consisting of a known number of cards, each of which has a type identifier. We then use a variety of rule-mining techniques applied with heuristic knowledge to attempt to predict the content of the deck after observing a specific number of cards chosen at random. It is important to note also that our game of choice is sufficiently complex, such that constructing a deck in the manner a human might is substantially more difficult than prediction using any method we have attempted here. Human players generally construct decks by identifying a central idea for the deck, then fitting cards into the deck that either support that concept or appeal to the player in some other way. While our techniques here produce similar results, there is no clear identification of concept, and all cards are connected, not selected for any other appeal.

This research could also be applied outside the realm of games, as this problem represents a highly complex, partially observable system with specific rules which govern the system construction. Optimising association rule mining to these complex requirements is clearly of interest as a general advancement of research in this area. The techniques here could easily be converted for use in other fields which have similar complex requirements on sets or multisets, simply by applying heuristic knowledge to data mining and rule generation processes as performed here.

The remainder of this paper is organised as follows. In section 2, we present a summary of related works on Rule Association Mining and other relevant techniques. Section 3 discusses the Android: Netrunner game which was the main focus for this work. In section 4, we discuss our experimental methods, the methods we used to generate association rules, and also the algorithms which we used to make deck predictions. In section 5, we present our results, and section 6 contains our conclusions and some notes on potential future work.

II. Related Work

The prediction of an opponent deck is effectively a form of opponent modelling [1], [2], [3], except with the important distinction that we are modelling strategic decisions which took place before the game started. As the opponent can't change their pre-game behaviour due to game experience, we do not need to create a full opponent model, only an estimation of actions which have already been performed. There has been little work in this specific area before, with the exception of a single application of machine learning to the game of Hearthstone [4], which achieved a very high prediction rate on a limited card set.

A. Association Rule Mining

Association Rule Mining is the determination of correlations between a set of items [5]. It is also known as *Market-Basket Analysis*, due to the common usage of determining which products a shopper may purchase based on what is already in their shopping basket. A typical rule-mining algorithm functions by generating rules that describe which items are likely to be included in a partially observed set,

[1]http://magic.wizards.com/
[2]http://us.battle.net/hearthstone/en/

given the items in the observable part of the set. Itemsets are drawn from the data such that each itemset describes a correlation between items. Association rule mining is employed in many application areas, including intrusion detection [6], web usage mining [7] and bioinformatics [8].

A commonly used algorithm in association rule mining is *Apriori* [9]. Apriori first generates all *1-itemsets* that appear in the data at least a number of times equal to a predetermined support value, then passes this generation onward to create a second generation of *2-itemsets*. This process continues until an empty generation is found (that is a generation with no candidates that appear at least *support* times in the data.) Each generation member then creates a single association rule which describe the correlation recognised by that member.

There are many variations on the Apriori technique to generate rules [10]. Most notable of these are a technique which attempts to identify the n-most interesting itemsets for rule generation rather than using a minimum support value [11], [12]. Some techniques also use functional languages rather than support constraints [13], and others use lattice and graphing techniques [14].

III. Android: Netrunner

Android: Netrunner is a two-player strategy card game published by Fantasy Flight Games[3], which includes elements of bluffing and deception. Netrunner is similar to other popular card games such as Magic:The Gathering, and is described as an LCG (Living Card Game [15]).

Due to the nature of the game, the content of an opponent's deck is critical strategy information, and a player who is able to accurately model their opponent's deck is at a substantial advantage. There are currently more than 600 cards released for Netrunner, so accurately modelling a deck is a significant challenge. The combination of the wide number of choices, plus the complex and specific rules for which cards may be included in decks makes Netrunner deck construction a highly intricate process.

During a standard match of Netrunner, opponents do not have access to the content of their opponents deck. Access to such information would provide a substantial advantage to a player, as they would both be able to predict their opponent's likely strategy, and also determine which strategies they are poorly defended against.

Netrunner has a well documented rules structure for deck building[4]. Each deck has a single identity card, which provides a *Side*, *Faction* and a certain amount of *Influence*. Decks may only include cards associated with their side, but may spend influence to include cards from other Factions.

Every Netrunner deck has exactly one *Identity* card which defines some rules for that deck, most notably a *Side*, an amount of *influence* and a *Faction*. There are exactly 2 sides (named *Runner* and *Corp*), and each card in Netrunner is associated with one side and cannot be included in decks associated with the other side. Identities which are from the corp side must also include a specific number of *agenda points*, which are provided corp cards (the specifics of agenda points are not relevant to this work, other than to recognise that there is a required number of agenda points for some decks to include, which presents an additional restriction upon decks.) All non-identity cards also have a *Faction* and a *Influence Cost*, the latter of which describes the amount of influence which must be paid to include the card in a deck which contains an identity of a different faction.

In this paper, we consider a deck for Netrunner to be a multiset, where no item can appear in a set more than three times. Each set also includes exactly one identity, which is always visible to us (as this is a condition of beginning a game of Netrunner), and also defines a portion of the multiset rules.

[3]http://www.fantasyflightgames.com

[4]https://images-cdn.fantasyflightgames.com/filer_public/2e/66/2e66279a-0b5c-4d12-80b1-754289b5ff0c/adn01_rules_eng_lo-res.pdf

IV. Experimentation Methods

A. Netrunner Deck Data

Experimentation data consisted of 6000 community made decklists posted on a popular Netrunner community website [5] that allows users to collect and compare decklists. Some filtered based on popularity was performed. Prediction accuracy results are determined by direct comparison of the predicted deck and the original deck and returning a percentage of the cards that match.

Algorithm 1 GetPredictedDeck(...) for a_1

function GetPredictedDeck(D_{obs}, R, C, n)

    ##Initialise all cards with rule support
    InitCardRuleCounts(D_{obs}, C, R)

    ##Sort cards desc by rule support
    sort(C, rulecount, 0)

    ##Set predicted deck to include observed deck
    $D_{pred} \leftarrow D_{obs}$

    ##For each card
    for all $c \in C$ **do**

        ##Take the required number of cards
        $k = min\{n - |D_{pred}|, c.MaxCount\}$

        ##Add them to the predicted deck, if possible
        D_{pred}.AppendMultiple(c,k)

B. Apriori Rule Generation

Rules were mined from data using the Apriori method detailed in Agrawal & Srikant [9], with modifications as detailed in sections below. This process generates a large number of *rules*, which describe the relationship between items in the analysed set. These rules are made up of one or more *antecedent* items, and one *consequent* item. The

[5]http://netrunnerdb.com

antecedent items is a multiset of items which must be found in any observed set in order for the rule to become active. The consequent item is the item which results from rule activation, and thus the item which will be added to the predicted set. Our rules take the form $\{A, B, B, C, D\} \rightarrow E$, where A, B, B, C, D is the full set of antecedents, and E is the consequent.

Each rule also has a *support* [16] value, which states how many occurrences of the complete set of antecedents and the consequent appear in the training data, and is useful to describe the magnitude of the effect of the rule. Support is calculated by the formula $support(X \rightarrow Y) = \sigma(X \cup Y)/N$ [17], where $(X \rightarrow Y)$ represents a rule, and N represents the total size of the data set. Each rule also has a confidence value, which measures the reliability of the rule. Confidence is calculated by the formula $confidence(X \rightarrow Y) = \sigma(X \cup Y)/\sigma(X)$.

The primary piece of evidence used to model an opponent's deck will be the identity card, as it is always visible, and also provides the constraints for deck construction in the form of faction, side and influence. As other cards are revealed through play, these can be added to the deck with complete confidence. It is usual to have observed a small number of opponent cards during the first turn of play (we estimate 1-4 is usual), and as such we vary the number of observed cards we randomly select to determine the effectiveness of our technique upon different sized sets of cards.

After rules were generated from the data, the set of 6000 decklists were tested using 30 fold cross-validation, with each individual prediction being made based upon a set of randomly selected cards from the decks. As these cards could potentially be duplicates, for each test a minimum of two unique cards are observed.

C. Apriori Prediction

1) Standard Apriori Prediction (a_1): The standard Apriori method of prediction is shown in algorithm 1, where D_{obs} represents the observed known cards, n represents the size of the observed deck, D_{pred} represents the predicted deck, R represents the set of all generated rules, and C represents the set of all Netrunner cards. In the first step of the algorithm we set the rule counts of each card to 0, then we run through all rules and determine if they are active for the set of cards we have observed (D_{obs}). We then set D_{pred} to contain D_{obs}, as our prediction will always include the cards we have observed, and this makes further operations easier. We sort all cards by their $rulecount$ attribute, and then move through them in decending order of $c.rulecount$ until we find sufficient cards to fill the remainder of D_{pred}.

2) Modifying for duplicate cards (a_2): A notable error performed by a_1 is number of duplicates which appear in the predicted decklists. As Netrunner decks can include up to three copies of each card[6], we attempt a technique that allows us to predict the number of copies of each item in the predicted multiset. Without this modification, the a_1 simply adds the maximum number of each item until it cannot add more, resulting in three copies of each card in the predicted deck.

Algorithm 2 GetPredictedDeck(...) for a_2

```
function GETPREDICTEDDECK(D_obs, R, C, n)

    ##Initialise all cards with rule support
    InitCardRuleCounts(D_obs, C, R)

    ##Sort cards desc by rule support
    sort(C, rulecount, 0)

    ##Set predicted deck to include observed deck
    D_pred ← D_obs

    ##For each card
    for all c ∈ C do

        ##Take the required number of cards
        k = min{n − |D_pred|, c.Cardinality}

        ##Add them to the predicted deck, if possible
        D_pred.AppendMultiple(c,k)
```

In order to modify this behaviour, we make a separate calculation using the rule metadata to determine the number of duplicates included in the original data. We then use this information to include copies in the prediction. This algorithm is very similar to algorithm 1 except that after a card is selected, the rule metadata is averaged to determine the number of duplicates to be included.

Therefore each run of $GetPrediction\_a_2(D_{obs})$ adds 1-3 cards to D_{obs}, and bans the included card from further selection. This technique may appear arbitrary, but in the case of duplication in a specific decklist, the nature of the individual card is far more relevant than any patterns between the card and other cards in that deck. For example, some cards are so strong and usable in any deck that they almost always appear in sets of 3, whereas others frequently appear alone due to the narrow field of use or difficulty to fit into a deck.

3) Prioritising by Influence (a_3): A review of the all data used here shows that 84% of decks in our dataset used all of their influence, 92% used all except 1 point, and 95% used all but 2. Considering that our data likely contains a large number of casual decks, which likely accounts for those not using all of the influence, this is indicative of how important the concern of influence during deck construction.

In order to prioritise influence spends, we change the method of deck prediction so that we first attempt to make predictions which would spend all available influence (both influence and non-influence cards still undergo the duplicate procedure mentioned in section IV-C2 above.) This new method is not shown in algorithm, as the only change is

[6] A few cards have specific rules which break this allow more copies or restrict the number of duplicates, but the vast majority may only appear in sets of 1-3

a sorting C so that all of the rules with a resultant card that will cost influence appear first, and this is restated later in algorithm 4. Notation is as before, however in the set C is sorted not only by rulecount, but also by a boolean that represents whether including any given card in D_{pred} would cost influence. This means that the first predictions made by a_3 will cost influence, and then when all the influence is spent, only cards that do not cost influence will be added.

4) Using influence during Rule Generation (a_4): Here, we separate item sequences that were generated from influence spend and non-influence spend. This allows us to separate the item sets into two groups, one which represents cards which players have spent influence on, and which represents card sequences that were used "in-faction". We can then generate specific rules for influence and non-influence spend. In the case that we had insufficient data, the prediction reverted to using all generated rules. This method is shown in algorithm 3. Notation is as before, however in addition R_{inf} represents rules originally generated from influence sets, and R_{noinf} represents rules which are generated from non-influence sets only. This algorithm is very similar to algorithm 2 except that $GetPrediction\_a_4$ uses only rules generated from influence selections when selecting an card that costs influence, and only rules generated from non-influence selections when selecting a card that doesn't cost influence.

5) Rule Generation including duplicate cards (a_5): We also attempted to remove the calculation for duplicate cards by allowing the rules to be constructed from duplicate items, and thus we should be able to predict those duplicates with more relevancy to the observed deck, rather than the general attributes of the cards. This algorithm is identical to algorithm a_4, except that duplicates are calculated based on the number of copies of each card seen in the generated rules rather than our cardinality data. When a rule is determined to be active, instead of checking rule metadata to determine the number of cards to add to the predicted deck, we instead determine the total number of the consequent item that already exist within the predicted deck, and if the required number specified by the rule already exist, we take no action. If the required number is not yet in the deck, we add a single consequent item. For example, if the rule $\{A, B, C\} \rightarrow B$ becomes active, we check to see if 2 or more B are included in the predicted deck. If so, we add nothing. If not, we add a single B.

6) Prioritising by rulesize (a_6): This modification attempts to give priority to rules which contain more items, as these rules will be less rarely active due to their specificity. However, when these rules are active for an observed card set, they will likely tell us more about the content of the deck than smaller rules. This algorithm is identical to a_4, except that the rules are sorted by descending rule size, and then a_4 is performed using the set of rules which are the largest size, then descending through the rules until we have completed the deck.

7) Making confident predictions (a_7): This modification is identical to a_6, however when we predict a card, we add it to

Algorithm 3 GetPredictedDeck(...) for a_4

```
function GETPREDICTEDDECK(D_obs, R_inf, R_noinf, C, n)

    ##Initialise all cards with rule support (inf)
    InitCardRuleCounts(D_obs, C, R_inf)

    ##Sort cards desc by rule support
    sort(C, rulecount, 0)

    ##Set predicted deck to include observed deck
    D_pred ← D_obs

    ##Spend influence first
    for all c ∈ C do

        ##Take the required number of cards
        k = min{⌊(maxinf − inf(D_pred))/c.inf⌋, c.Cardinality}

        ##Add them to the predicted deck, if possible
        D_pred.AppendMultiple(c,k)

    ##Initialise all cards with rule support (no inf)
    InitCardRuleCounts(D_obs, C, R_noinf)

    ##Then fill the deck with non-influence cards
    for all c ∈ C do

        ##Take the required number of cards
        k = min{n − |D_pred|, c.Cardinality}

        ##Add them to the predicted deck, if possible
        D_pred.AppendMultiple(c,k)
```

the observed card set and check all rules again. So any card we predict to appear in the deck, we assume we are correct for the purposes of further predictions. This final version is shown in algorithm 4.

V. Results

All results for predictions are shown in figure 3. Use of the mentioned techniques to generate deck predictions is generally successful, completing decks with an accuracy of up to 59% from viewing only 5 cards (roughly 8-10% of the actual deck). However there are some general trends which can be observed. Firstly, as each card (or set of cards) are added to the deck sequentially, we don't take into account new patterns which may emerge between originally observed cards and cards more recently added. This means that all predictions are based on the original set of observed cards, whereas we would likely have a different effect on prediction if we considered predicted cards to be part of the observed set when making further predictions. We suggest that some of the difference in prediction may be a tendency to form into familiar deck archetypes, as predicted cards would likely support larger patterns already recognised as frequently

Algorithm 4 GetPredictedDeck(...) for a_7

```
function GETPREDICTEDDECK(D_obs, R_inf, R_noinf, C, n)
    D_pred ← D_obs
    for all r ∈ R_inf do

        ##Initialise all cards with inf rule support
        InitCardRuleCounts(D_pred, C, R_inf)

        ##Sort cards desc by rule support
        sort(C, rulecount, 0)

        ##Spend influence first
        for all c ∈ C do

            ##Take the required number of cards
            k = min{⌊(maxinf − inf(D_pred))/c.inf⌋, c.Cardinality}

            ##Add them to the predicted deck, if possible
            D_pred.AppendMultiple(c,k)
        for all r ∈ R_noinf do

            ##Initialise all cards with non-inf rule support
            InitCardRuleCounts(D_pred, C, R_noninf)

            ##Sort cards desc by rule support
            sort(C, rulecount, 0)

            ##Fill out deck with non-influence
            for all c ∈ C do

                ##Take the required number of cards
                k = min{n − |D_pred|, c.Cardinality}

                ##Add them to the predicted deck, if possible
                D_pred.AppendMultiple(c,k)
    return D_pred
```

played decks. This is somewhat consistent with human deck construction however, as players often use existing archetypes to construct decks.

In order to provide a control for experimentation, random selection was tested (a_0). Generated decks were still required to observe deck construction rules, but other than that cards were selected randomly from the set of available cards. All predictions using a_0 had an accuracy in the range 0% - 6%, and due to this low accuracy, results are not shown below.

We also attempted to test prediction across a range of different numbers of observed cards. In each of these cases, the identity card was always observed, then an additional number of cards were added. This means in the case of the number of observed cards being zero, only the identity card was observed. In all previous experiments the size of the set has been five, which represents what a player might expect from two complete turns of play. We tested prediction with sets of up to ten viewed cards. We also tested prediction with a set of zero observed cards, which represents the game before play has begun.

A. *Default Apriori (a_1)*

Default Apriori allows for predictions of up to 48% accuracy, and while this is somewhat effective, it can be improved upon significantly by the later algorithms which incorporate heuristic knowledge. Different values of minimum support were used to determine the optimum value, which lies close to 15. All of these tests were run on a dataset of size 200 (30-fold cross-validation on a total set of size 6000), so larger values of minimum support will likely cause smaller detail of the dataset to be lost during rule generation. Examination of the decks generated with a_1 also reveals that almost every card is included in triplets, further speaking of the necessity of a modification to address the number of duplicates included.

B. *Apriori with duplicates (a_2)*

The modification to consider inclusion of duplicates in the predicted deck results in a significant increase in accuracy. The most significant value of minimum support now appears between 10-15, both options resulting in a prediction accuracy of 53%, an increase in accuracy of 5%. This increase in accuracy is certainly related to more accurate predictions on sets of duplicate cards, as due to the nature of the game, certain cards are more often played in sets of 2 or 3, and certain cards are almost always played without duplicates. This modification largely makes the effect that there are no longer automatic inclusions of cards in groups of 3, however it can still be further improved with respect to heuristic data.

C. *Apriori with Influence Priority (a_3)*

While prioritising the inclusion of cards which cost influence has a positive effect, the effect is marginal, increasing prediction accuracy by less than ∼2% at the optimal value of minimum support 10. It is surprising that the effect is so marginal, but upon examining further it is apparent that most (92%) of decks predicted with a_1 and a_2 already include the maximum permitted influence for those decks, so the modification is perhaps not as important to prediction as originally proposed.

Examinations of the individual card selections shows that the influence spends are somewhat inappropriate however, and are somewhat to blame for the inaccuracies of this prediction algorithm.

D. *Apriori with Influence Filtering (a_4)*

There are several interesting effects in these results. Firstly, the highest accuracy has risen to 57%, an increase of ∼4%. Secondly, the optimal value of minimum support has changed to a higher value of 20.

A review of the cards selected by influence spends reveals that they are much more appropriate to the acknowledged deck archetypes, presumably due to the specific use of rules generated entirely from influence spend patterns.

We also start to observe some occasional single-card influence inclusions which are well established in the appropriate archetypes.

E. Rule Generation including duplicate cards (a_5)

We can see from the results for a_5 that attempting to determine the number of duplicate cards in a deck from generated rules appears to be less effective than using our data on the normal set count of that card. This is believable, as the number of duplicate cards included is likely to be much more dependent on the nature of the card than on the nature of the deck itself. As our information relates to patterns between cards, we don't necessarily have a good understanding of the nature of the card itself.

It is worth noting however that for some values of minimum support, a_5 is approximately as effective as a_3 and a_2, meaning that it is still an effective technique, and alternative methods to predict duplicate cards in the deck could be investigated.

F. Prioritising by rulesize (a_6)

Giving priority to larger rules has also had a positive effect on prediction accuracy. We can see this effect particularly when minimum support is 20. We attribute this effect to larger rules being more rarely satisfied unless they are highly informative about the configuration of decks. As such, activated large rules should be given priority over activated smaller rules.

G. Making confident predictions (a_7)

By adding all predictions to our observed set, we are assuming that all our predictions are correct, and biasing future predictions by this information. This has a positive effect on prediction accuracy at higher values of minimum support, however it has almost no effect at values of 15 and below. This could be explained by some subtly of rules that are activated with a support of 15 or less, however in this case we would expect the prediction accuracy to be positively affected also, and yet we see that this is not the case.

The extension of our observed set also has another less obvious effect on prediction, which is that it allows activation of rules with larger item sequences, as more items appear in the observed set. This means as D_{obs} expands, we may observe decks activating larger rules, and effectively falling into archetypes.

H. Varied Size Observation Set

The results for predictions made with varied observation sets are shown in figure 1. We can see that the overall change in deck prediction accuracy across the total range of tested values is approximately 20%, which while a large change, might be less than we expect from such a change in source data. This illustrates the importance of the identity card which is always viewed, it speaks deeply of the construction of the deck, mostly because the identity card is always active during play, and a substantial portion of the cards included will have some synergy with that identity. This also speaks of the nature of deck construction in Netrunner, which largely consists of modifications to existing archetypes, likely due to smaller synergies between groups of cards. It is also worth noting that at almost all values of observed set size and minimum support, our algorithms which incorporate heuristic information perform significantly better than default apriori.

We see an understandable increase in prediction accuracy as we increase the size of the observed set, as there are both fewer cards to predict, and also more information is available on the set content. Rules with a higher number of antecedents are also activated, which likely provides more accurate information on the set content.

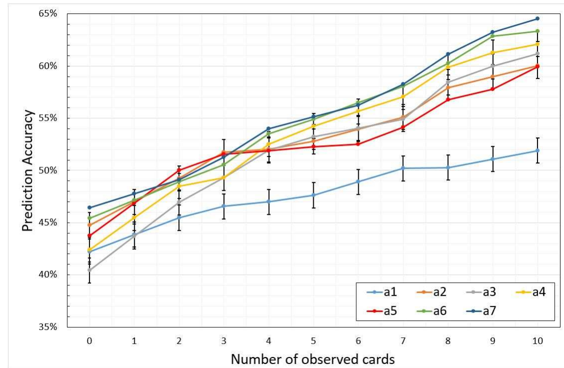

Fig. 1. Varied Size Observation Set

We can also observe that a few of our own techniques (a_3&a_4) perform very poorly when the observed set is very small or empty. As a_3 and a_4 both focus on influence inclusions, this is likely due to a lack of corroborating information from other observed cards to distinguish correct influence selections. As such, the initial influence selections are almost unguided, and as these cards are selected from a much larger set of available cards than regular selections, the picks are more likely to be incorrect without guidance.

There is also an interesting plateau in prediction accuracy around set size 3-6 with algorithm a_5. This is likely due to the estimation for duplicate cards struggling on smaller set size. As the cards in the observed section of the deck are selected randomly during each test, it is possible that duplicate cards are selected, and as such less information is exposed in certain cases. This might cause a decrease in accuracy when only a small number of unique cards are observed. This calculation is not included in any other algorithm, as it was not effective in increasing accuracy overall, possibly due to this complication.

The results across all experiments grouped by algorithm are shown in figure 2. We can more clearly see a general rise in prediction accuracy here, with the exception of the a_5 algorithm for reasons mentioned above. This is to be expected, as each algorithm following a_1 includes specific heuristic improvements which are targeted to improve efficiency in this specific domain.

Algorithm a_5 shows that our introduction of rule-based cardinality estimations have been unsuccessful in improving prediction efficiency, although this is something we would definitely want to address in future. The current cardinality estimations are unlikely to predict decks with 100% accuracy, for example it will always fail to predict decks that include a unusually small number of a card almost always seen in sets of 3.

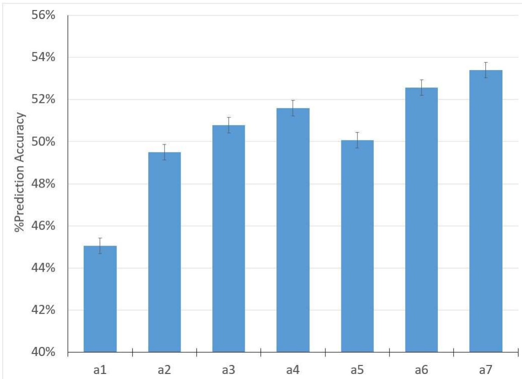

Fig. 2. Cumulative results by Algorithm

VI. Conclusions & Further Work

It can be seen that our modifications to the Apriori technique provide a significant improvement to prediction of decks in Netrunner, showing a maximum improvement of ∼13% between the default apriori algorithm (a_1) and our optimal modified algorithm (a_7).

There are several other opportunities for future work which could be explored. For example, the technique used to separate rules in a_4 could also be applied to identity cards, using only rules generated for each identity to select either the entire deck, or the influence-spend portion of the deck. However this would require a large amount of data, as certain identities are unpopular and may appear only rarely within our current data set, so there would be fewer useful rules generated for these identities. It may also be worth looking at generation by *Faction*, which might yield more interesting results. Also, as our observed cards were randomly selected, they may not adequately represent the real order cards are observed during a game (as it is more common to play certain cards earlier than others.) Biasing generation of the observed set of card may provide a more realistic scenario.

Our attempts to adequately predict the number of duplicate cards within a deck have been some what effective, but there is still work to be done here, as our best prediction is based on our heuristic knowledge of the specific card, rather than knowledge of the card in context. Successfully adding contextual heuristic knowledge into this process will surely lead to more accurate prediction.

We can also look to applying these techniques to other domains, specifically the games mentioned in section 1. Magic the Gathering has a much larger set of active cards, and less stringent deck construction rules, so while this would represent a more challenging target, there is also a much larger amount of data available due to the larger player community and history of the game. Hearthstone likely represents a point of medium complexity, as the card pool is between the other two games mentioned here (approximately 450), and the deck construction rules are more restrictive than Magic, and thus provide more guidance.

A further avenue of research which could be pursue is that of pattern matching within the decks, in order to draw out common patterns which occur within multiple decks, and then using that information to further bias the prediction.

Acknowledgements

The work displayed here was supported by EPSRC (http://www.epsrc.ac.uk/), the LSCITS program at the University of York (http://lscits.cs.bris.ac.uk/), the NEMOG program at the University of York (http://www.nemog.org/), and Stainless Games Ltd (http://www.stainlessgames.com/).

References

[1] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner, "Bayes' Bluff: Opponent Modelling in Poker," in *Proceedings of the TwentyFirst Conference on Uncertaintyin Artificial Intelligence UAI*, 2005, pp. 550–558.

[2] M. Ponsen, G. Gerritsen, and G. M. J.-B. Chaslot, "Integrating Opponent Models with Monte-Carlo Tree Search in Poker," in *Proc. Conf. Assoc. Adv. Artif. Intell.: Inter. Decis. Theory Game Theory Workshop*, no. February, 2010, pp. 37–42.

[3] C. Bauckhage, C. Thurau, and G. Sagerer, "Learning human-like opponent behavior for interactive computer games," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2781, pp. 148–155, 2003. [Online]. Available: http://www.cs.berkeley.edu/daf/games/webpage/AIpapers/Bauckhage2003-LHL.pdf

[4] C. Bursztein and E. Bursztein, "I am a legend: Hacking Hearthstone with machine learning," in *DEFCON 22*, 2014, p. 169. [Online]. Available: https://cdn.elie.net/talks/I-am-a-legend-defcon-22-slides-final.pdf

[5] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Record*, vol. 22, no. May, pp. 207–216, 1993.

[6] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, 1999, pp. 120–132.

[7] J. Srivastava, R. Cooley, M. Deshpande, and P.-n. Tan, "Web usage mining: discovery and applications of usage patterns from Web data," vol. 1, no. 2, pp. 12–23, 2000.

[8] C. Creighton and S. Hanash, "BIOINFORMATICS Mining gene expression databases for association rules," pp. 79–86, 2003.

[9] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," *Proceeding VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases*, vol. 1215, pp. 487–499, 1994.

[10] J. Hipp, U. Güntzer, and G. Nakhaeizadeh, "Algorithms for association rule mining — a general survey and comparison," *ACM SIGKDD Explorations Newsletter*, vol. 2, no. 1, pp. 58–64, 2000.

[11] S. C. Ngan, T. Lam, R. C. W. Wong, and A. W. C. Fu, "Mining N-most interesting itemsets without support threshold by the COFI-tree," *International Journal of Business Intelligence and Data Mining*, vol. 1, no. 1, p. 88, 2005. [Online]. Available: http://www.inderscience.com/link.php?id=7320

[12] A. W. C. Fu, R. W.-w. Kwong, and J. Tang, "Mining N-most Interesting Itemsets," in *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems (ISMIS),*, 2000.

[13] Z. Hu, W. Chin, and M. Takeichi, "Calculating a new data mining algorithm for market basket analysis," *Practical aspects of declarative languages: second International Workshop, PADL 2000, Boston, MA, USA, January 17-18, 2000: proceedings*, pp. 169–185, 2000.

[14] J. Hipp, A. Myka, R. Wirth, and U. Guntzer, "A New Algorithm for Faster Mining of Generalized Association Rules," in *Principles of Data Mining and Knowledge Discovery*, 2006, pp. 74–82.

[15] S. C. Duncan, "Mandatory Upgrades: The Evolving Mechanics and Theme of Android: Netrunner," in *Well Played Summit*, 2014.

[16] V. Baez-Monroy and S. O'Keefe, "An Associative Memory for Association Rule Mining," *2007 International Joint Conference on Neural Networks*, no. 2, pp. 2227–2232, 2007.

[17] P.-N. Tan, M. Steinbach, and V. Kumar, "Association Analysis: Basic Concepts and Algorithms," *Introduction to Data mining*, pp. 327–414, 2005. [Online]. Available: http://www-users.cs.umn.edu/{∼}kumar/dmbook/index.php

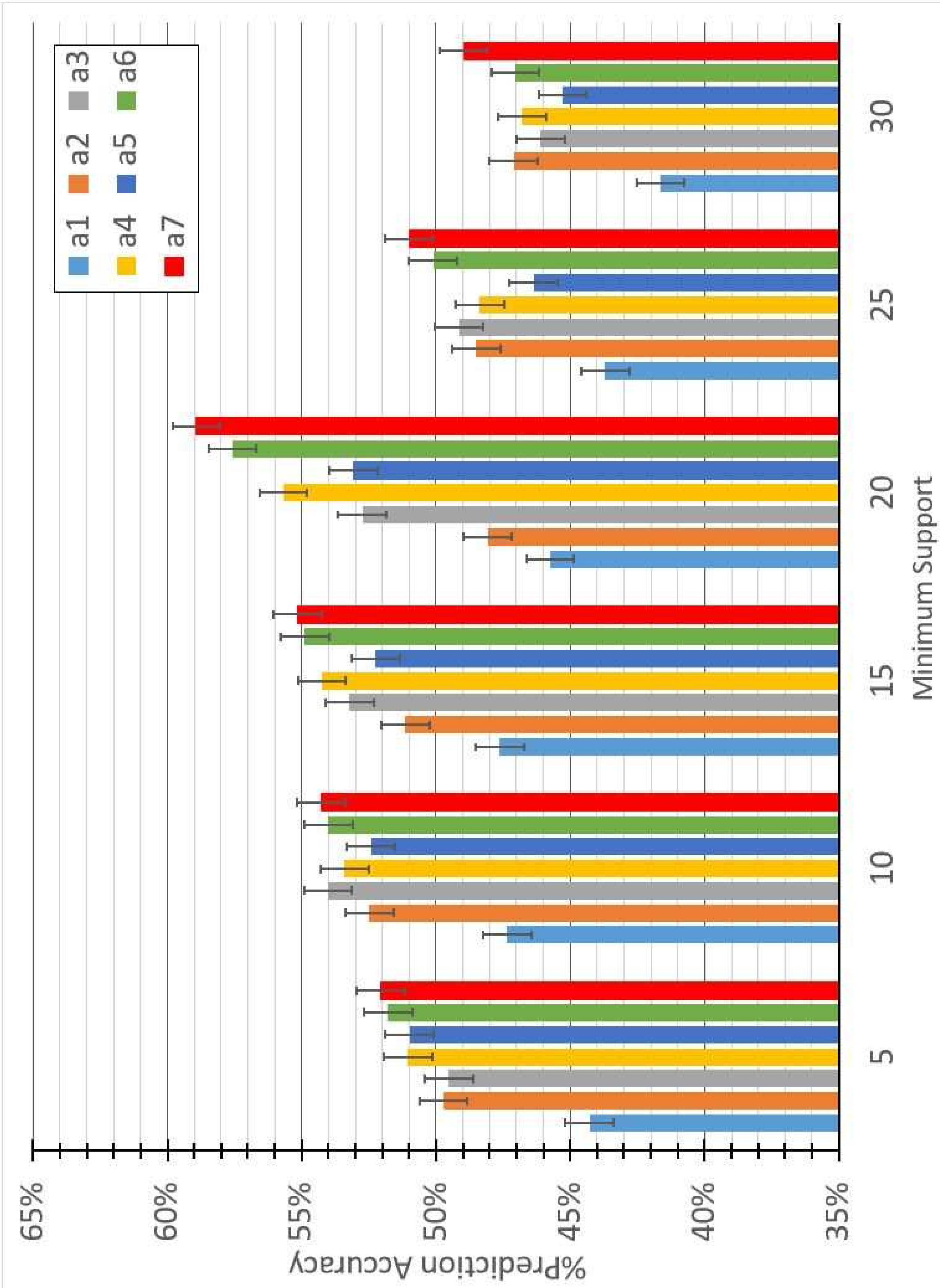

Fig. 3. Results of algorithm runs with varying minimum support values